Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей
Кафедра электронных вычислительных машин
Дисциплина: Операционные системы и системное программирование

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовому проекту
на тему

«МНОГОПОТОЧНАЯ ПРОГРАММА ДЛЯ ФОНОВОГО КОНТРОЛЯ ИЗМЕНЕНИЙ И ЦЕЛОСТНОСТИ ГРУППЫ ФАЙЛОВ»

БГУИР КР 1-40 02 01 101 ПЗ

Студент М.В. Антимонов

Руководитель Л.П. Поденок

Минск 2025

Учреждение образования
«Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

УТВЕРЖДАЮ
Заведующий кафедрой
(подпись)
_____ 2025 г.

ЗАДАНИЕ
по курсовому проектированию

Студенту Антимонову Максиму Владиславовичу

1. Тема проекта: *Многопоточная программа для фонового контроля изменений и целостности группы файлов*

2. Срок сдачи студентом законченного проекта: *15 мая 2025 г.*

3. Исходные данные к проекту: *язык программирования C*

4. Содержание расчетно-пояснительной записки (перечень вопросов, которые подлежат разработке):

Введение. 1. Обзор методов и алгоритмов решения поставленной задачи. 2. Обоснование выбранных методов и алгоритмов. 3. Описание программы для программиста. 4. Описание алгоритмов решения задачи. 5. Руководство пользователя. Заключение. Список использованных источников. Приложения.

5. Перечень графического материала (с точными обозначениями обязательных чертежей и графиков):

1. *Схема алгоритма работы функции*
2. *Скриншоты работы программы*
3. *Ведомость документов*

6. Консультант по проекту (с обозначением разделов проекта) *Поденок Л. П.*

7. Календарный график работы над проектом на весь период проектирования (с обозначением сроков выполнения и трудоемкости отдельных этапов):

раздел 1 к 01.03. – 15%;

раздел 2, 3 к 01.04. – 50%;

раздел 4, 5 к 01.05. – 80%;

оформление пояснительной записки и графического материала к 15.05.2025 – 100%

Защита курсового проекта с 29.05 по 09.06.

Руководитель Л.П. Поденок

Задание принял к исполнению М.В. Антимонов

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Многопоточная программа мониторинга изменений и проверки целостности файлов предназначена для обеспечения контроля за состоянием критически важных файлов в системе. В условиях повышенных требований к безопасности и целостности данных своевременное обнаружение несанкционированных или непредвиденных изменений в файлах становится необходимым компонентом комплексной защиты информационных систем.

Разработанная программа представляет собой многопоточное приложение для операционных систем на базе `Linux`, которое обеспечивает постоянный мониторинг заданной группы файлов с использованием как активных уведомлений через механизм inotify, так и периодического опроса с помощью отдельных потоков. Программа вычисляет криптографические хеши файлов для обнаружения даже самых незначительных изменений в их содержимом, что позволяет выявлять модификации, которые могут остаться незамеченными при использовании только стандартных атрибутов файловой системы (время модификации, размер).

Основные возможности программы включают:

1) Мониторинг модификаций, удалений и переименований файлов в реальном времени;

2) Рекурсивное сканирование директорий для полного охвата файловой иерархии;

3) Параллельную обработку множества файлов с использованием многопоточной архитектуры;

4) Вычисление и сравнение криптографических хешей (`SHA-256`) для надежной проверки целостности;

5) Гибкую систему логирования событий с возможностью использования системного журнала (`syslog`);

6) Автоматическое обнаружение и мониторинг новых файлов в отслеживаемых директориях.

Программа предназначена для применения в критических инфраструктурах, где необходим постоянный контроль за состоянием конфигурационных файлов, исполняемых модулей и других важных элементов системы.

1 ОБЗОР МЕТОДОВ И АЛГОРИТМОВ РЕШЕНИЯ ПОСТАВЛЕННОЙ ЗАДАЧИ

1.1 Методы мониторинга изменений в файловой системе

Для мониторинга изменений в файловой системе современные ОС предоставляют специальные механизмы событийного типа. В отличие от устаревшего подхода с периодическим опросом, такие решения работают через подписку на события ядра, что обеспечивает мгновенное уведомление о любых изменениях. Данная архитектура позволяет добиться минимальной задержки обнаружения изменений при оптимальном использовании системных ресурсов. Наиболее распространённые реализации такого подхода включают:

1) Механизм `inotify` представляет собой `API` ядра `Linux`, который позволяет получать уведомления о событиях файловой системы через файловый дескриптор. `Inotify` уведомляет приложение о таких событиях как открытие, закрытие, модификация, удаление, перемещение файла или изменение атрибутов. Механизм работает на уровне ядра и потребляет значительно меньше ресурсов по сравнению с периодическим опросом;

2) Механизм `fanotify` более новый `API`, который расширяет функциональность `inotify`, добавляя возможность контроля доступа к файлам и мониторинга на уровне монтирования, а не только отдельных файлов или директорий. `Fanotify` обладает более высокими привилегиями, но требует соответствующих прав доступа;

3) Механизм `kqueue` используется в BSD-подобных системах для мониторинга файловых дескрипторов и генерирует события при изменениях файлов. Не доступен в стандартных дистрибутивах Linux.

Также существуют и альтернативные подходы, основанные на периодическом опросе (`polling`), которые не зависят от специфики ОС, но требуют больше ресурсов и работают с задержкой:

1) Периодическое сканирование – метод, при котором система вручную проверяет атрибуты файлов (время изменения, размер) или их содержимое. Универсален, но ресурсоемок и обнаруживает изменения с задержкой, зависящей от частоты сканирования;

2) Отложенная проверка по расписанию – вариант периодического сканирования, запускаемый через `cron`. Подходит для аудита и задач, где не требуется мгновенная реакция, но не обеспечивает мониторинг в реальном времени.

1.2 Алгоритмы проверки целостности файлов

Для обеспечения достоверности данных и выявления изменений в содержимом файлов применяются различные методы контроля целостности, которые можно разделить на две категории: криптографические хеш-функции и некриптографические методы проверки. Каждый из этих подходов имеет свои особенности, область применения и уровень надежности.

Криптографические хеш-алгоритмы представляют собой наиболее надежный способ проверки целостности данных, обеспечивая высокую степень защиты от преднамеренных изменений. Среди наиболее распространенных алгоритмов можно выделить:

1) MD5 (Message Digest 5) – это быстрый алгоритм хеширования, генерирующий 128-битный хеш. Несмотря на свою скорость, MD5 обладает известными уязвимостями к коллизиям, что делает его непригодным для серьезных криптографических применений. Тем не менее, он продолжает использоваться в случаях, когда требуется быстрая проверка целостности в некритичных с точки зрения безопасности сценариях;

2) SHA-1 (Secure Hash Algorithm 1) – алгоритм создающий 160-битный хеш, долгое время считался стандартом в криптографии. Однако с обнаружением уязвимостей к коллизиям его использование в новых приложениях было прекращено. В настоящее время SHA-1 сохраняет ограниченное применение в унаследованных системах;

3) SHA-256/SHA-512 (Secure Hash Algorithm 2) – более современные алгоритмы, входящие в семейство SHA-2, являются текущим стандартом криптографической защиты. SHA-256 генерирует 256-битный хеш, а SHA-512 –512-битный. Они обеспечивают высокий уровень безопасности и широко применяются для проверки целостности файлов, цифровых подписей и других криптографических операций;

4) SHA-3 (Secure Hash Algorithm 3) – новейший стандарт криптографического хеширования, основанный на алгоритме Keccak. Обеспечивает высокую степень безопасности, но используется реже из-за более позднего внедрения.

Для менее критичных задач, где не требуется криптографическая стойкость, могут применяться более простые методы контроля целостности:

1) CRC32 (Cyclic Redundancy Check) – быстрый алгоритм обнаружения случайных ошибок в данных. Хотя он не обеспечивает защиты от преднамеренных изменений, его эффективность для выявления случайных искажений делает CRC32 популярным решением в сетевых протоколах и

системах хранения данных;

2) Сравнение метаданных файлов – метод, основанный на анализе таких атрибутов, как размер файла, время последней модификации и права доступа. Главное преимущество этого подхода – отсутствие необходимости читать и обрабатывать содержимое файла, что значительно ускоряет проверку.

1.3 Архитектуры механизма параллельной обработки

Для эффективного мониторинга большого количества файлов необходимо задействовать параллельные вычисления, которые позволяют распределить нагрузку и ускорить обработку данных. В зависимости от масштаба задачи можно выделить следующие модели многопоточной обработки:

1) `Thread per task` – модель, которая подразумевает создание отдельного потока для каждой задачи. Простая модель, но неэффективная при большом количестве файлов из-за накладных расходов на создание и управление потоками;

2) `Thread pool` – заранее созданный набор потоков, который используется для выполнения задач. Более эффективное использование ресурсов за счет уменьшения накладных расходов на создание и уничтожение потоков;

3) `Work stealing` – модель, в которой потоки, завершившие свои задачи, могут «воровать» ожидающие задачи у других потоков. Это особенно полезно при неравномерной загрузке, так как позволяет динамически балансировать работу между всеми доступными вычислительными ресурсами.

Поскольку потоки работают с общими ресурсами, критически важно предотвращать состояния гонки и гарантировать согласованность данных. Для этого используются следующие примитивы

1) Мьютексы (`Mutual Exclusion`) обеспечивают эксклюзивный доступ к ресурсу, позволяя только одному потоку выполнять операцию в конкретный момент времени;

2) Условные переменные (`Condition Variables`) позволяют потокам приостанавливать выполнение до наступления определенного условия, экономя вычислительные ресурсы.

2 ОБОСНОВАНИЕ ВЫБРАННЫХ МЕТОДОВ И АЛГОРИТМОВ

2.1 Выбор методов мониторинга

Для реализации программы мониторинга файлов были выбраны два дополняющих друг друга метода.

Активный мониторинг на основе `inotify`. Выбор `inotify` обусловлен его эффективностью и низким потреблением ресурсов по сравнению с периодическим опросом. `Inotify` позволяет получать мгновенные уведомления о событиях файловой системы, что критически важно для быстрого реагирования на изменения. Механизм `inotify` интегрирован в ядро `Linux`, что обеспечивает его высокую стабильность и производительность.

Преимущества выбора `inotify`:

- Минимальная задержка обнаружения изменений;
- Низкое потребление ресурсов процессора и системы в целом;
- Подробная информация о типе события (создание, модификация, удаление, перемещение).

Резервный мониторинг на основе периодического опроса. В качестве дополнительного механизма выбран периодический опрос, что обеспечивает надежность системы в случае ограничений `inotify` (например, при достижении лимита наблюдаемых файлов) или при его недоступности. Этот подход также позволяет обнаруживать изменения, которые могли произойти во время инициализации системы мониторинга.

Обоснование комбинированного подхода:

- Резервирование критически важной функциональности;
- Возможность работы в системах с ограниченной поддержкой `inotify`;
- Обнаружение изменений, произошедших в период неактивности мониторинга.

2.2 Выбор алгоритма проверки целостности

Для обеспечения надежной проверки целостности файлов выбран криптографический алгоритм хеширования `SHA-256`, реализуемый через библиотеку `OpenSSL.`

Алгоритм SHA-256 представляет оптимальный баланс между криптографической стойкостью и производительностью. По сравнению с MD5 и SHA-1, он обеспечивает более высокую защиту от коллизий, что критически важно для надежного обнаружения модификаций в файлах.

Обоснование выбора SHA-256:

- Криптографическая стойкость, достаточная для задач мониторинга целостности;

- Широкая поддержка в криптографических библиотеках;

- Разумные требования к вычислительным ресурсам по сравнению с более тяжелыми алгоритмами (SHA-512, SHA-3);

- Отсутствие известных практических атак, в отличие от MD5 и SHA-1.

Для реализации алгоритма SHA-256 выбрана библиотека OpenSSL, что обусловлено её надежностью, оптимизированной реализацией и широким распространением в Linux-системах.

Преимущества использования OpenSSL:

- Высокоавтоматизированная реализация, включающая ассемблерные оптимизации для различных архитектур;

- Обширное тестирование и регулярные обновления безопасности;

- Стандартизированный API для работы с криптографическими примитивами.

2.3 Выбор многопоточной архитектуры

Для обеспечения эффективного мониторинга большого количества файлов выбрана многопоточная архитектура на основе POSIX Threads (pthread) с комбинацией подходов:

Специализированные потоки для отдельных функций. Выделение отдельного потока для обработки событий inotify позволяет быстро реагировать на изменения в файловой системе без блокировки основного потока приложения.

Пул рабочих потоков для мониторинга файлов. Создание пула потоков, количество которых настраивается в конфигурации, для параллельной обработки файлов при периодическом опросе. Каждый поток обрабатывает подмножество контролируемых файлов, что обеспечивает более эффективное использование многоядерных процессоров.

Обоснование выбранной архитектуры:

- Масштабируемость: количество потоков может быть адаптировано к доступным вычислительным ресурсам;

- Независимость работы механизмов мониторинга: `inotify` и периодический опрос работают параллельно;

- Эффективное использование многоядерных процессоров для ускорения проверки больших наборов файлов.

Для предотвращения условий гонки и обеспечения корректной работы с общими данными выбраны следующие механизмы синхронизации:

- Мьютексы для защиты доступа к общему списку файлов;

- Условные переменные для сигнализации об изменениях между потоками.

Данный выбор обеспечивает надежную синхронизацию доступа к разделяемым ресурсам с минимальными накладными расходами.

2.4 Обоснование дополнительных проектных решений

1) Реализована гибкая система логирования с поддержкой различных уровней детализации и возможностью вывода как в стандартный поток, так и в системный журнал (`syslog`) или файл. Это решение обеспечивает удобство диагностики проблем и анализа работы программы в различных сценариях использования;

2) Параметры работы программы вынесены в отдельный конфигурационный файл, что позволяет адаптировать поведение программы без повторного комплирования. Такой подход повышает гибкость и удобство использования системы;

3) Реализован механизм отслеживания перемещения файлов с использованием "`cookie`" событий `inotify`, что позволяет корректно обрабатывать переименования и перемещения файлов без потери информации о них;

4) Поддержка рекурсивного сканирования директорий позволяет мониторить целые иерархии файлов, что существенно упрощает настройку программы при работе с большими наборами файлов.

Совокупность выбранных технических решений обеспечивает создание эффективной, надежной и гибкой системы мониторинга целостности файлов, способной адаптироваться к различным условиям эксплуатации и требованиям безопасности.

3 ОПИСАНИЕ ПРОГРАММЫ ДЛЯ ПРОГРАММИСТА

Основная реализация программы мониторинга файлов выполнена в двух ключевых компонентах:

- `file_monitor.c` – основной файл, содержащий логику работы программы;

- `file_monitor.conf` – задает параметры работы мониторинга;

- `Makefile` – обеспечивает процесс сборки программы.

Файл `file_monitor.c` содержит основные функциональные блоки для реализации работы программы.

Функция `calculate_hash()` вычисляет `SHA-256` хеш файла с использованием `OpenSSL`.

Передаваемые параметры:

1) `const char *filepath` – путь к файлу для хеширования;

2) `unsigned char *hash` – буфер для записи полученного хеша.

Функция `hash_to_string()` преобразует бинарный хеш в строковое шестнадцатеричное представление.

Передаваемые параметры:

1) `unsigned char *hash` – указатель на бинарный хеш;

2) `char *string` – буфер для записи строкового представления хеша.

Функция `init_file_info()` инициализирует структуру информации о файле.

Передаваемые параметры:

1) `const char *filepath` – путь к файлу;

2) `file_info_t *file_info` – указатель на структуру для инициализации.

Функция `check_file_changes()` проверяет, были ли изменения в файле.

Передаваемые параметры:

1) `file_info_t *file_info` – указатель на информацию о файле.

Функция `add_file()` добавляет файл в список отслеживаемых.

Передаваемые параметры:

1) `const char *filepath` – путь к файлу для добавления.

Функция check_file_existence() проверяет существование файла и

обновляет список при удалении.

Передаваемые параметры:

1) `int index` – индекс файла в массиве `files`.

Функция `scan_directory()` рекурсивно сканирует директорию и добавляет файлы для мониторинга.

Передаваемые параметры:

1) `const char *dirpath` – путь к директории для сканирования.

Функция `init_inotify()` инициализирует систему уведомлений `inotify`.

Функция `find_dir_by_wd()` находит путь к директории по ее `watch`-дескриптору.

Передаваемые параметры:

1) `int wd` – `watch`-дескриптор `inotify`.

Функция `time_string()` возвращает текущее время в строковом формате.

Функция `log_message()` записывает сообщение в лог с указанным уровнем важности.

Передаваемые параметры:

1) `int level` – уровень важности сообщения;

2) `const char *format` – формат сообщения;

3) `...` – переменное число аргументов для форматирования.

Функция `read_config()` читает конфигурацию программы из файла.

Передаваемые параметры:

1) `const char *config_path` – путь к конфигурационному файлу.

Функция `monitor_thread()` – функция потока мониторинга с использованием периодического опроса.

Передаваемые параметры:

1) `void arg` – идентификатор поток.

Функция `inotify_thread()` – функция потока мониторинга с использованием `inotify`.

Функция handle_signal() – обработчик сигналов для корректного завершения программы.

Передаваемые параметры:

1) `int sig` – полученный сигнал.

Функция `cleanup()` освобождает выделенные ресурсы перед завершением программы.

Функция `main()` – точка входа в программу. Функция обрабатывает параметры командной строки, читает конфигурацию, инициализирует отслеживание файлов, создает потоки мониторинга и координирует их работу до получения сигнала завершения.

Передаваемые параметры:

1) `int argc` – количество аргументов командной строки;

2) `char *argv[]` – массив строк-аргументов.

Файл `file_monitor.conf` содержит следующие настраиваемые параметры:

1) `check_interval` – интервал проверки файлов при использовании `polling`-режима;

2) `thread_count` – количество рабочих потоков мониторинга;

3) `use_inotify` – выбор механизма мониторинга;

4) `recursive_scan` – рекурсивное сканирование поддиректорий;

5) `watch_new_files` – автоматическое добавление новых файлов;

6) `use_syslog` – интеграция с системным логгером;

7) `log_level` – выбор уровня логирования;

8) `log_file` – путь к файлу для логирования.

Основная логика реализована в `file_monitor.c`, который включает функции для хеширования, проверки изменений, работы с `inotify`, многопоточной обработки и логирования. Конфигурация программы задается через `file_monitor.conf`, позволяя настраивать режимы работы, параметры потоков и уровни детализации логов. Сборка и управление программой осуществляются через `Makefile`, обеспечивая удобство развертывания.

4 ОПИСАНИЕ АЛГОРИТМОВ РЕШЕНИЯ ЗАДАЧИ

5 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ